OBSERVANT: an Annotated Term Rewriting System for Deciding Observation

Observant: an Annotated Term-Rewriting System for Deciding Observation Congruence*

Raúl Monroy
Departamento de Ciencias de la Computación, ITESM – Campus Estado de México
Apdo. Postal 50, Módulo de Servicio Postal Campus Edo. de México del ITESM
52926 Atizapán, Edo. de México
raulm@campus.cem.itesm.mx

Alan Bundy and **Ian Green**
Department of Artificial Intelligence, Edinburgh University
80 South Bridge, EH1 1HN, Edinburgh, United Kingdom
bundy, img@dai.ed.ac.uk

Abstract

The use of term-rewriting systems (TRSs) is at the core of automated theorem proving (ATP). There are a number of methods whereby we can build terminating TRSs with full deduction power, with respect to some theory. Yet, there exist theories these techniques cannot handle, giving rise to a trade-off between deduction power and termination. One such theory is the theory of observation congruence [Milner, 1989b] in CCS [Milner, 1989a]. The standard approach to overcome this situation is to use meta-level information; it, however, introduces the problem of finding and capturing such meta-level control. This paper advocates the use of annotated TRSs (ATRSs) for approaching these problems; the annotations are used to capture meta-level information that helps direct proof search, while giving an intuitive account about why and how rewriting is expected to work. We introduce an ATRS, called Observant. Observant is goal-oriented, and, hence, domain-specific: it can be used to build a decision procedure for observation congruence in CCS [Milner, 1989a]; furthermore, it can be used to attempt to solve or prove CCS equations, involving either finite-state systems, or infinite-state systems. We prove Observant terminating, sound, and complete – over the normalisation of finite processes. We argue that Observant surpasses rival term-rewriting strategies.

keywords: term-rewriting systems, annotated TRSs, CCS, process normalisation.

1 Introduction

The use of term-rewriting systems (TRSs) is at the core of automated theorem proving. TRSs are chiefly applied to simplify algebraic expressions, serving the basis for efficient computational techniques.

Nice properties of many sets of rewrite rules include *termination* and *confluence*. A TRS is terminating if there is no infinite sequence of rewriting steps in it. A TRS is confluent if whenever[1] $s \overset{*}{\Leftarrow} t \overset{*}{\Rightarrow} q$, then there exists a term t' such that $s \overset{*}{\Rightarrow} t' \overset{*}{\Leftarrow} q$ [Dershowitz, 1989]; moreover, a TRS is *canonical* if it is both terminating and confluent.

Confluent TRSs also satisfy the nice property of being *Church-Rosser*: a TRS $\mathcal{R}$ produced from an equational theory Γ is Church-Rosser if any two expressions that can be shown equal with the aid of Γ can also be shown equal by developing their respective $\mathcal{R}$-rewriting search trees, until a common expression is uncovered. If $\mathcal{R}$ is also terminating and finite, then $\mathcal{R}$ constitutes a decision procedure over the equality of two terms in Γ.

Some theories give rise to no canonical TRS. On the one hand, termination may fail to hold when the TRS includes axioms like commutativity $X + Y \Rightarrow Y + X$, or when two rules come from an equation used both ways round, e.g. $X + (Y + Z) \Rightarrow (X + Y) + Z$ and $(X + Y) + Z \Rightarrow X + (Y + Z)$. On the other hand, confluence may fail to hold when the theory includes operators enjoying either idempotence or absorption, both of the schematic form: $X = T(X)$. While there is currently a well-established repertoire of techniques to derive terminating TRSs, e.g. associative path orderings (apo) [Bachmair and Plaisted, 1985], the standard completion strategies, such as those based on critical pairs, show limitations to provide confluence. This is because some theories give rise to infinitely many critical pairs that do not reduce to the identity relation [Inverardi and

*The research reported in this paper was supported by EPSRC grant GR/L/11724 and CONACyT studentship 64745.

[1] The symbol $\Rightarrow$ denotes a term-rewriting relation; furthermore, let $\mathcal{R}$ be a relation, then $\mathcal{R}^*$ stands for the transitive, reflexive closure of $\mathcal{R}$.

Nesi, 1995]. One such theory is the theory of observation congruence [Milner, 1989b] in CCS [Milner, 1989a].

Experience has shown that, often, it is possible to use meta-level information to overcome the above problem. So, [Dershowitz, 1989] has suggested to extend the rewriting rule of inference with a meta-level applicability condition. This approach, however, exhibits three main disadvantages. First, inefficiency: rewrite-rule application depends on global conditions; the overall state must be therefore tested, updated and validated. Second, specialisation: the output TRS is geared towards a theory, constraining its applicability in other domains. Third, specialists are required for discovering, formalising, and mechanising the meta-level control information. STRAT [Inverardi and Nesi, 1995], a canonical TRS for normalising CCS terms in the theory of observation congruence, makes use of this model.

Another approach to capture meta-level control in TRSs is the use of annotations. An annotated TRS is a TRS in which the (object-level) terms are provided with (meta-level) decorations. The aim of the annotations are twofold: i) to give an account as to why and how the ATRS should work (*expectancy*), and ii) to direct the search for a solution, since matching includes annotations (*efficiency*). ATRSs are, thus, goal-oriented: they often involve little search; moreover, due to the expectancy property, it is possible to exploit failure in proof attempts. Rippling [Bundy *et al.*, 1993; Basin and Walsh, 1996] is one such annotated TRS; it has been successfully and extensively applied in several domains, including automated inductive theorem proving [Bundy, 1988; Bundy *et al.*, 1991], summing series [Walsh *et al.*, 1992], limit theorems [Yoshida *et al.*, 1994], correcting faulty conjectures [Monroy *et al.*, 1994], etc. This range of applications might provide evidence that lifting meta-level control information to annotations can extend the applicability of a search procedure (*generality*); hence, avoiding the Second problem shown by the Dershowitz Rewriting approach.

This paper reports on an experiment about the use of ATRSs: we introduce an ATRS, called `Observant`, to control proof search in the theory of observation congruence. `Observant` can be made to play a chief role in both process normalisation (over observation congruence), and equation solving (over various behavioural classes, including infinite state processes). Here, we will see that rewriting with `Observant` is terminating and carefully guided, pruning significantly the proof search space. Furthermore, we will see how `Observant` can be used to build a decision procedure for process normalisation, over observation congruence.

The rest of the paper is organised as follows: In §2, we discuss CCS and the theory of observation congruence; there, we shall look into the absorption lemma, a key result for process normalisation. §3 presents `Observant`, while §4 shows its properties. In §5, we compare our work, draw conclusions, and discuss further work.

2 CCS

Terms of CCS represent processes; processes have their own identity, circumscribed by their entire capabilities of interaction with both other agents, and the environment. Such interactions are nothing else but communicating activity; they are simply referred to as *actions*. An action is said to be *observable*, if it denotes an interaction between one agent and its environment; otherwise, it is said to be *silent*, or *unobservable*. This interpretation of observation underlies a precise and amenable theory of behaviour: whatever is observable is regarded as the behaviour of a system; two agents are concluded equivalent, if their behaviour is indistinguishable to an external observer.

Syntax The set of actions, $\mathcal{A}ct$, ranged over by meta-variables $\alpha, \beta, \ldots$, contains names (e.g. in), co-names (e.g. $\overline{in}$) and the unobservable action τ, which denotes internal communication. The set of processes, $\mathcal{P}$, ranged over by meta-variables $P, Q, \ldots$, contains expressions with the following syntax:

$$P ::= \mathbf{0} \mid \alpha.P \mid P + P$$

$\mathbf{0}$ denotes the deadlock process, capable of no actions whatever. The combinator *Prefix*, $(.)$, is used to convey the discrete actions an agent may perform; for example, $\alpha.P$ denotes a process capable of executing the action α, and then evolving into the process P. The combinator *Summation*, $(+)$, disjoins the capabilities of the agents that appear either side of it; as soon as one performs any action, the other is dismissed.

Semantics CCS is given meaning by means of the labelled transition system $(\mathcal{P}, \mathcal{A}ct, \{\xrightarrow{\alpha} : \alpha \in \mathcal{A}ct\})$, where $\xrightarrow{\alpha}$ is the smallest transition relation satisfying the rules below:

Act $\alpha.P \xrightarrow{\alpha} P$, for every $\alpha \in \mathcal{A}ct$; and

Sum If $P_1 \xrightarrow{\alpha} P'$ or $P_2 \xrightarrow{\alpha} P'$ then $P_1 + P_2 \xrightarrow{\alpha} P'$

Process Equivalence The experimental idea used as a basis for process analysis can be stated, informally, as follows:

> P and Q are equivalent if, after abstracting any internal process communication, they can match each other's actions, and they evolve into equivalent processes.

This property is characterised in the definition of *bisimulation* [Park, 1981]. [Milner, 1989a] defines bisimulation as follows:

Definition 1 (bisimulation) *Let $\mathcal{S}$ be a binary relation over processes; then $\mathcal{S}$ is a* bisimulation *if $(P, Q) \in \mathcal{S}$ implies, for all $\alpha \in Act$,*

i) Whenever $P \xrightarrow{\alpha} P'$ and $\alpha \neq \tau$ then, for some Q', $Q \stackrel{\alpha}{\Rightarrow} Q'$ and $(P', Q') \in \mathcal{S}$, and similarly for P and Q interchanged;

ii) Whenever $P \xrightarrow{\tau} P'$ then either, for some Q', $Q \stackrel{\tau}{\Rightarrow} Q'$ and $(P', Q') \in \mathcal{S}$, or $(P', Q) \in \mathcal{S}$, and similarly for P and Q interchanged

where $P \stackrel{\alpha}{\Rightarrow} P'$ *means* $P(\stackrel{\tau}{\rightarrow})^* \stackrel{\alpha}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* P'$. *Whenever* $P \stackrel{\alpha}{\Rightarrow} P'$, P' *is called an* α-descendant *of* P.

Then, observation equivalence is defined to be the union of all weak bisimulations:

Definition 2 (Bisimilarity) P *and* Q *are* observation equivalent *or (*weakly*)* bisimilar, *written* $P \approx Q$, *if* $(P, Q) \in \mathcal{S}$ *for some (weak) bisimulation* $\mathcal{S}$.

Observation equivalence is not a congruence relation; that is, in general, we cannot replace a term for another observation equivalent term, while preserving equivalence. The largest congruence relation included in $\approx$ is called *observation congruence* $\approx^c$. $\approx^c$ can be characterised either as a bisimulation [Milner, 1989a], or as the maximum congruence defined by $\mathcal{A}$, the axiom system below[2]. $\mathcal{A}$ has been proved sound and complete for $\approx^c$, i.e. $P \approx^c Q$ iff $\mathcal{A} \vdash P = Q$ [Hennessy and Milner, 1985].

Definition 3 (The Axiom System $\mathcal{A}$)

A1: $P + Q = Q + P$
A2: $P + (Q + R) = (P + Q) + R$
A3: $P + P = P$
A4: $P + \mathbf{0} = P$
A5: $\alpha.\tau.P = \alpha.P$
A6: $\tau.P + P = \tau.P$
A7: $\alpha.(\tau.Q + P) + \alpha.Q = \alpha.(\tau.Q + P)$

2.1 Deciding Observation Congruence

When proving the completeness of $\mathcal{A}$, [Hennessy and Milner, 1985] hinted how to construct a method for deciding observation congruence, based on process normalisation:

> P and Q are observation congruent if their corresponding normal forms are identical (syntactically speaking), modulo the associativity and commutativity of $+$.

Roughly speaking, the normal form of a process corresponds to a process which contains no summand that can be "absorbed" by another; put in another way, normal form means zero redundancies, in terms of process summands and external aspects of behaviour. For example, $P + P$ is not a normal form, as $P + P = P$. More precisely,

Definition 4 (Normal Form) P *is a* sumform *if it is of the form* $\sum_{i=1}^{n} \mu_i.P_i$, *where each* P_i *is a sumform. Notice that, in particular,* $\mathbf{0}$ *is a sumform, by taking* $n = 0$.
A sumform $P \equiv \sum_{i=1}^{n} \mu_i.P_i$ *is a* proper normal form *if i) it is not of the form* $\tau.P'$, *ii) each* P_i *is a proper normal form; and iii) for* $i \neq j$, P_i *is not congruent to any* μ_i*-descendant of* $\mu_j.P_j$; *that is, it is* not *the case that* $P_i \stackrel{\mu_j}{\Rightarrow} P_j$, *for* $j \neq i$.
$\tau.P$ *is an* improper normal form, *if* P *is a proper normal form.*
A normal form *is either a proper or an improper normal form.*

The key condition in the definition of proper normal form is iii); it states the absorption above mentioned, and formally given by the following lemma [Hennessy and Milner, 1985]:

Lemma 1 (absorption) *Let* P *be a sumform, then*

$$P \stackrel{\alpha}{\Rightarrow} Q \text{ implies } \mathcal{A} \vdash P + \alpha.Q = P.$$

Whenever $P \stackrel{\alpha}{\Rightarrow} Q$, we call P *solvent*; reciprocally, we call $\alpha.Q$ the *excess* or *irrelevant summand*. Naturally, by the transition semantics, i.e. the rules **Act** and **Sum**, $P \stackrel{\alpha}{\Rightarrow} Q$ implies both $(P + \mu.R) \stackrel{\alpha}{\Rightarrow} Q$, and $(\mu.R + P) \stackrel{\alpha}{\Rightarrow} Q$. Whenever $P \stackrel{\alpha}{\Rightarrow} Q$ and P is just a single process, i.e. not of the form $P_1 + P_2$, we call P *absorbent*. Notice that solvents contain one and only one absorbent.

Thus, by means of absorption, we can eliminate irrelevant summands; moreover, if applied recursively on the depth of the term structure – starting from the deepest levels, absorption obtains the normal form of the input term (nf) [Hennessy and Milner, 1985]. That is

$$\begin{aligned}
&\mathrm{nf}(P) = \\
&\quad \textbf{if } P \equiv \mathbf{0} \textbf{ then } \mathbf{0}; \\
&\quad \textbf{if } P \equiv \alpha.P' \textbf{ then } \texttt{Observant}^\star(\alpha.\mathrm{nf}(P')) \\
&\quad \textbf{if } P \equiv (\sum_{i=1}^{n} P_i) \textbf{ then } \texttt{Observant}^\star(\sum_{i=1}^{n} \mathrm{nf}(P_i))
\end{aligned}$$

this normalisation procedure already hints at this paper's ultimate goal: Given $P, \alpha.Q \in \mathcal{P}$, such that $P \stackrel{\alpha}{\Rightarrow} Q$, drive the use of $\mathcal{A}$, by means of the rewrite-rule of inference, to absorb $\alpha.Q$ into P.

$\mathcal{A}$ does not give rise to a canonical TRS for process normalisation; the superposition of the rules below, derived from **A3**, **A6** and **A7**, gives rise to an infinite number of critical pairs, that do not reduce to the identity.

$$\begin{aligned}
P + P &:\Rightarrow P \\
\tau.P + P &:\Rightarrow \tau.P \\
\alpha.(\tau.Q + P) + \alpha.P &:\Rightarrow \alpha.(\tau.Q + P)
\end{aligned}$$

Moreover, process normalisation involves rewriting with the following rules:

$$\begin{aligned}
&r1\text{:} \quad P :\Rightarrow P + P \\
&r2\text{:} \quad \tau.P :\Rightarrow \tau.P + P \\
&r3\text{:} \quad \alpha.(\tau.Q + P) :\Rightarrow \alpha.(\tau.Q + P) + \alpha.P
\end{aligned}$$

and, hence, is essentially non-terminating.

To see why the use of *r1*, *r2*, and *r3* is mandatory, consider the following proof search example. To prove that $\tau.(P + Q) + P = \tau.(P + Q)$, we have to "saturate" the term $\tau.(P+Q)$, using $r2$, leaving $\tau.(P+Q)+(P+Q)+P = \tau.(P+Q)$; then "re-group" the term $(P+Q)+P$ to $(P+P)+Q$, leaving $\tau.(P+Q)+((P+P)+Q) = \tau.(P+Q)$; then "absorb" P into P yielding $\tau.(P+Q)+(P+Q) = \tau.(P+Q)$; and, finally, absorb $P + Q$ into $\tau.(P + Q)$, using the rule $\tau.P + P :\Rightarrow \tau.P$.

These subtleties will be all handled by `Observant`, which we introduce below.

3 `Observant`: The Rewriting Strategy

3.1 Annotations

`Observant` operates on a special kind of annotated terms; these are as follows:

[2] We omit the standard rules for reflexivity, symmetry and substitutivity of equality.

1. unannotated terms are `Observant` terms,
2. Whenever $P \stackrel{\alpha}{\Rightarrow} Q$ and we wish to rewrite $P + \alpha.Q$ into P, then
   - the excess, $\alpha.Q$, is marked with a dotted circle, $\overset{\cdots}{(\alpha.P)}$
   - the solvent, P, is marked with solid circles, using this procedure, written in Prolog:

     $\texttt{ann}(\alpha.P, \alpha.Q, \overset{\alpha}{\bigcirc}.P') \texttt{ :- ann0}(P, Q, P').$
     $\texttt{ann}(\tau.P, \alpha.Q, \tau.P') \texttt{ :- ann}(P, \alpha.Q, P').$
     $\texttt{ann}(P_1 + P_2, \alpha.Q, P') \texttt{ :-}$
     $\quad \texttt{ann}(P_1, \alpha.Q, P'); \texttt{ ann}(P_2, \alpha.Q, P').$

     $\texttt{ann0}(P, P, \overset{P}{\bigcirc}).$
     $\texttt{ann0}(\tau.P, Q, P') \texttt{ :- ann0}(P, Q, P')$
     $\texttt{ann0}(P_1 + P_2, Q, P') \texttt{ :-}$
     $\quad \texttt{ann0}(P_1, Q, P'); \texttt{ ann0}(P_2, Q, P').$

     Notice that, whenever $P \stackrel{\alpha}{\Rightarrow} Q$, `ann/3` annotates P so that

$$P \overset{\alpha}{\Rightarrow} \overset{Q}{\bigcirc}$$

3. annotated expressions yielded by `Observant` rewritings are `Observant` expressions
4. only annotated expressions formed using these rules are `Observant` expressions

3.2 Annotated Rewrite-Rules

`Observant` uses a special kind of annotated rewrite-rules. These rules are classified according to the task they perform: tidying/merging, attraction, saturation and absorption, and applied following this order, until no rule of any class is applicable. After each rule application, this order is restarted; this is the so-called *waterfall approach*. The rules are all applied using a call-by-value search strategy.

Tidying/Merging
Tidying simply eliminates internal τ actions:

$$\overset{\alpha}{\bigcirc}.\tau.\overset{P}{\bigcirc} \;:\Rightarrow\; \overset{\alpha.P}{\bigcirc}$$

while merging glues the annotated term $\overset{\alpha}{\bigcirc}.\overset{Q}{\bigcirc}$ into $\overset{\alpha.Q}{\bigcirc}$. Merging is a pure meta-level step, i.e. it corresponds to no object-level proof step.

Attraction
Attraction moves the excess next to the absorbent[3]. It applies the following rules, which assume that the absorbent appears before the excess, when the expression is read left to right[4]:

$$(\, C_1(\overset{P}{\bigcirc}) + Q \,) + C_2(\overset{\cdots}{R})$$
$$:\Rightarrow\; (\, C_1(\overset{P}{\bigcirc}) + C_2(\overset{\cdots}{R}) \,) + Q$$
$$(\, Q + C_1(\overset{P}{\bigcirc}) \,) + C_2(\overset{\cdots}{R})$$
$$:\Rightarrow\; Q + (\, C_1(\overset{P}{\bigcirc}) + C_2(\overset{\cdots}{R}) \,)$$
$$C_1(\overset{P}{\bigcirc}) + (Q + C_2(\overset{\cdots}{R}))$$
$$:\Rightarrow\; (\, C_1(\overset{P}{\bigcirc}) + C_2(\overset{\cdots}{R}) \,) + Q$$
$$C_1(\overset{P}{\bigcirc}) + (C_2(\overset{\cdots}{R}) + Q)$$
$$:\Rightarrow\; (\, C_1(\overset{P}{\bigcirc}) + C_2(\overset{\cdots}{R}) \,) + Q$$

Saturation
Saturation makes progress towards the absorption of the excess, while keeping track of the absorbent movements[5].

$$\tau.C(\overset{P}{\bigcirc}) + \overset{\cdots}{Q} :\Rightarrow$$
$$\lfloor \tau.C(P) \rfloor + C(\overset{P}{\bigcirc}) + \overset{\cdots}{Q}$$
$$\overset{\alpha}{\bigcirc}.(\tau.C(\overset{Q}{\bigcirc}) + P) + \overset{\cdots}{R} :\Rightarrow$$
$$\lfloor \alpha.(\tau.C(Q) + P) \rfloor + \overset{\alpha}{\bigcirc}.C(\overset{Q}{\bigcirc}) + \overset{\cdots}{R}$$

Notice that the terms onto which saturation rules are applied are marked with this piece of notation $\lfloor \ldots \rfloor$; such annotations gives us the means of reversing any transformation to obtain the original term.

Absorption
The aim of the absorption rules is twofold, according to the sort of annotation. First, it absorbs the excess into the absorbent:

$$\overset{P}{\bigcirc} + \overset{\cdots}{P} \;:\Rightarrow\; P$$
$$\tau.\overset{P}{\bigcirc} + \overset{\cdots}{P} \;:\Rightarrow\; \tau.P$$
$$\overset{\alpha}{\bigcirc}.(\tau.\overset{Q}{\bigcirc} + P) + \overset{\cdots}{\alpha.Q} \;:\Rightarrow\; \alpha.(\tau.Q + P)$$

Second, it undoes previous saturation steps, if any, by getting rid of the absorbent:

$$\lfloor \tau.P \rfloor + P \;:\Rightarrow\; \tau.P$$
$$\lfloor \alpha.(\tau.Q + P) \rfloor + \alpha.Q \;:\Rightarrow\; \alpha.(\tau.Q + P)$$

Procedurally, the steps tidying/merging, attraction, absorption, and saturation are interpreted as follows:

- After tidying/merging, use attraction to move the excess close to the summand that contains the absorbent subcomponents. Then apply absorption.
- If absorption is successful, then terminate;
- Otherwise, proceed to saturate; that is, pop a copy of the absorbent up the term structure into the next level, while keeping track of this movement. Saturation will eventually enable absorption. In that case, absorption will get rid of the excess and apply the saturation steps in reverse order, using the tracks left by saturation.

[3] N.B. the absorbent is static.
[4] C_i ($i \in \{1, 2\}$) stands for a second-order variable.
[5] C is other than the identity function.

An example is illustrative. Consider that we wish to absorb $\mu.R$ into $\tau.(\tau.Q + \mu.R)$. The absorption is possible, since, by operational semantics, we know that $\tau.(\tau.Q + \mu.R) \stackrel{\mu}{\Rightarrow} R$. Take the following context[6]:

$$\ldots + \tau.(\tau.Q + \mu.R) + \mu.R + \ldots$$

`ann/3` annotates this expression, yielding:

$$\ldots + \tau.(\tau.Q + \textcircled{\mu.R}) + \dddot{\mu.R} + \ldots$$

Attraction is inapplicable, because the excess and the absorbent are next to each other; absorption is also inapplicable, because the expression does not match any of the absorption rules. Only saturation can be used, it yields

$$\ldots + (\lfloor \tau.(\tau.Q + \mu.R) \rfloor + (\tau.Q + \textcircled{\mu.R})) + \dddot{\mu.R} + \ldots$$

Then, the waterfall starts again. With this formula, the use of attraction is suggested, with which the expression is left as follows:

$$\ldots + \lfloor \tau.(\tau.Q + \mu.R) \rfloor + (\tau.Q + (\textcircled{\mu.R} + \dddot{\mu.R})) + \ldots$$

Because attraction moves the excess close to the absorbent, the structure of the saturated term is preserved; this will allow us to go back along the path marked by saturation.

The absorption rules will complete the job now. First, the excess is removed:

$$\ldots + \lfloor \tau.(\tau.Q + \mu.R) \rfloor + (\tau.Q + \mu.R) + \ldots$$

then, the saturation step is undone, leaving:

$$\ldots + \tau.(\tau.Q + \mu.R) + \ldots$$

just as required.

4 Properties of `Observant`

4.1 Termination

We analyse each rewriting class first.

Proposition 1 *Tidying terminates*

Proof. Trivial. □

Proposition 2 *absorption terminates*

Proof. The measure is the number of annotated summands. On application, each rule eliminates both a summand, and the annotations of the absorbent; thence absorption is measure decreasing. □

Proposition 3 *attraction terminates*

Proof. As in PRESS [Sterling *et al.*, 1982], the measure for attraction rules is the distance between the terms $C(\textcircled{P})$ and $\dddot{R}$, defined as the length of the shortest path between the terms – regarding the tree associated to the structure of the expression containing them. For example, the distance between $C(\textcircled{P})$ and $\dddot{R}$, in the LHS of each attraction rule, is 3, while it is 2 in the corresponding RHS. Thus, each rule is measure decreasing. □

To prove that saturation terminates, we will need the notions below:

Definition 5 (Saturable term and level) *A process P is said to be* saturable *if and only if it matches any of the (saturation) rules r1, r2 and r3 (see §2.1). The* saturation level *of P is defined inductively as follows:*

$$\begin{aligned} \text{sat\_level}(\mathbf{0}) &= 0 \\ \text{sat\_level}(\alpha.P) &= \textit{saturable}(\alpha.P) + \text{sat\_level}(P) \\ \text{sat\_level}(P + Q) &= \text{sat\_level}(P) + \text{sat\_level}(Q) \end{aligned}$$

where $\textit{saturable}(\alpha.P)$ returns 1 if $\alpha.P$ is saturable, and 0 otherwise.

Definition 6 (Saturation order) P_1 *is* less saturable *than P_2, if the level of saturation of P_1 is less than the level of saturation of P_2.*

Proposition 4 *saturation terminates*

Proof. The annotations in these rules avoid the saturation of an already saturated term. Moreover, the absorbent produced by these rules is less saturable than the solvent. So, saturation is applicable a finite number of times. □

Theorem 1 `Observant` *terminates*

Proof. No rewriting step can increase the measure of absorption and saturation. Absorption preserves the measure of attraction; however, saturation can increase it. Both because, by Proposition 4, the absorbent is moved up the term structure, into the level where the excess is, in a finite number of steps, and because no rewriting step introduces any further excess, it follows that attraction is used only a finite number of times. Then, by Proposition 3, each time that attraction is used, it terminates. Finally, since, by Proposition 2, absorption is applied finitely many times, it follows that `Observant` terminates. □

4.2 Soundness and Completeness

Theorem 2 `Observant` *is sound: whenever* `Observant` *rewrites $P + \alpha.Q$ into P, then $P + \alpha.Q = P$*

Proof. Trivial: annotated rewrite-rules are decorated object-level rewrite rules, each preserves equivalence; so, it follows that $P + \alpha.Q :\Rightarrow^* P$ implies $P + \alpha.Q = P$.

Theorem 3 `Observant` *is complete: whenever $P \stackrel{\alpha}{\Rightarrow} Q$, then* `Observant` *is guaranteed to rewrite $P + \alpha.Q$ into P*

Proof. As in [Hennessy and Milner, 1985], we proceed by induction on the structure of an α-descendant, i.e. $P \stackrel{\alpha}{\Rightarrow} Q$ iff $P(\stackrel{\tau}{\rightarrow})^* \stackrel{\alpha}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* Q$; so, we consider three cases:

Case i) or base case Prove that $P \stackrel{\alpha}{\rightarrow} Q$ implies $P + \alpha.Q = P$. Since P is a sumform, then, by the transition semantics, $P \stackrel{\alpha}{\rightarrow} Q$ implies that P is of the form: $\underbrace{S + \alpha.Q + T}_{P}$, where S and T stand for some

[6] Henceforth, ellipsis will be used to denote expressions that remain unchanged.

other, possibly empty, context. In that case, we prove that

$$\vdash \underbrace{S+\alpha.Q+T}_{P}+\alpha.Q=\underbrace{S+\alpha.Q+T}_{P}$$

using `Observant` as follows: first, we use `ann/3` to identify and annotate the absorbent, $\alpha.Q$, leaving:

$$\vdash \underbrace{S+\textcircled{\alpha.Q}+T}_{P}+\alpha.Q=\underbrace{S+\alpha.Q+T}_{P}$$

second, we mark the excess:

$$\vdash S+\textcircled{\alpha.Q}+T+\dddot{\alpha.Q}=\ldots$$

third, using the attraction rules, we move the excess next to the absorbent:

$$\vdash S+(\textcircled{\alpha.Q}+\dddot{\alpha.Q})+T=\ldots$$

and fourth, using the absorption rules, we eliminate the excess: the result follows:

$$\vdash S+\alpha.Q+T=S+\alpha.Q+T$$

Notice how the whole sequence of term transformations has the good quality of leaving the syntactic structure of the solvent, P, unchanged.

Case ii) Take

$$P' \stackrel{\tau}{\Rightarrow} Q \text{ then } P'+\tau.Q :\Rightarrow^* P' \qquad (1)$$

to be the induction hypothesis, and

$$P \stackrel{\alpha}{\rightarrow}\stackrel{\tau}{\Rightarrow} Q \text{ then } P+\alpha.Q :\Rightarrow^* P$$

to be the induction conclusion. By the transition rules, $P \stackrel{\alpha}{\rightarrow} P'$ means that P is of the form $S+\alpha.P'+T$; then, by (1),

$$S+\alpha.P'+T=S+\alpha.(P'+\tau.Q)+T$$

then, to prove $P+\alpha.Q=P$, using `Observant`, we proceed as follows: first, using `ann/3`, we annotate both the solvent, $P \stackrel{\textcircled{\alpha}}{\rightarrow}\stackrel{\tau}{\Rightarrow} \textcircled{Q}$, and the excess, $\dddot{\alpha.Q}$:

$$\vdash S+\textcircled{\alpha}.(P'+\tau.\textcircled{Q})+T+\dddot{\alpha.Q}= S+\alpha.(P'+\tau.Q)+T$$

second, as in the basis case, we move the excess closed to the absorbent:

$$\vdash \ldots+\textcircled{\alpha}.(P'+\tau.\textcircled{Q})+\dddot{\alpha.Q}+\ldots=\ldots$$

third, we apply the absorption rules to absorb $\alpha.Q$ into $\alpha.(P'+\tau.Q)$. So, the result follows:

$$\vdash \ldots+\alpha.(P'+\tau.Q)+\ldots= \ldots+\alpha.(P'+\tau.Q)+\ldots$$

Case iii) Taking

$$P' \stackrel{\alpha}{\Rightarrow} Q \text{ then } P'+\alpha.Q :\Rightarrow^* P' \qquad (2)$$

to be the induction hypothesis; we prove the induction conclusion

$$P \stackrel{\tau}{\rightarrow}\stackrel{\alpha}{\Rightarrow} Q \text{ then } P+\alpha.Q :\Rightarrow^* P$$

So again, by transition semantics, $P \stackrel{\tau}{\rightarrow} P'$ implies that P is of the form $S+\tau.P'+T$, and in that case $P=S+\tau.(P'+\alpha.Q)+T$ by hypothesis, (2). Then, to prove $P+\alpha.Q=P$, using `Observant`, we annotate both the solvent, $P \stackrel{\tau}{\rightarrow}\stackrel{\textcircled{\alpha}}{\Rightarrow} \textcircled{Q}$, and the excess, $\dddot{\alpha.Q}$:

$$\vdash S+\tau.(P'+\textcircled{\alpha.Q})+T+\dddot{\alpha.Q}= S+\tau.(P'+\alpha.Q)+T$$

then, using the attraction rules, we move the excess and the absorbent together

$$\underbrace{\ldots+\tau.(P'+\textcircled{\alpha.Q})+\dddot{\alpha.Q}}_{Rd}+\ldots=\ldots$$

and then, we do the following rewritings onto Rd:

$$(\lfloor\tau.(P'+\alpha.Q)\rfloor+(P'+\textcircled{\alpha.Q}))+\dddot{\alpha.Q}=\ldots$$

By saturation rules

$$\lfloor\tau.(P'+\alpha.Q)\rfloor+(P'+\alpha.Q)=\ldots$$

By attraction/absorption rules, as in case i)

$$\tau.(P'+\alpha.Q)=\ldots$$

By absorption rules

In both case ii) and case iii), we have made the assumption that the expressions all meet the following conditions: a) the absorbent appears, when read left to right, first; b) the absorbent is either of the forms:

$$\textcircled{\alpha}.C(\textcircled{Q})+\ldots \qquad (3)$$

$$\tau.(D(\textcircled{\alpha}.F(\textcircled{Q}))+\ldots)+\ldots \qquad (4)$$

$C(\textcircled{Q})$ is either $\textcircled{Q}$, or $\tau.(C'(\textcircled{Q})+\ldots)$; similarly, $C'(\textcircled{Q})$ is either $\textcircled{Q}$, or $\tau.(C''(\textcircled{Q})+\ldots)$, and so on. $D(\textcircled{\alpha}.F(\textcircled{Q}))$ is either of the form (3), or of the form (4). These forms are easy to produce; furthermore, they are preserved by `Observant` rewriting. Thence, it follows that if $P \stackrel{\alpha}{\Rightarrow} Q$, and if P is of any of the above forms, then by means of `Observant` we can transform $P+\alpha.Q$ into P. □

5 Comparison, Conclusions, and Further Work

An application of the absorption lemma can be automatically translated into a sequence of rewrite-rule applications. This is evidenced by `Observant`. Terms are only saturated if by that means the absorbent is popped out so as to, eventually, perform an absorption. Thus, it never over-saturates; this is in contrast with STRAT, which uses an apo – as opposed to an absorbability order – to carry out saturations, hence increasing the complexity of the proof search method.

`Observant` is both sound and complete with respect to the absorption lemma; hence, it can be used for process normalisation; however, unlike STRAT, it can also be used in the context of equation solving, involving finite state agents, infinite state agents, and parameterised agents. This is because `Observant` requires only the knowledge of the excess and the absorbent to perform, $P \stackrel{\alpha}{\Rightarrow} Q$; its behavious does not depend on the syntax, but the associated transition semantics. This is in contrast with STRAT, which needs the input term to be in $\mathcal{P}$. `Observant` therefore surpasses STRAT.

The system R_OBS [Inverardi and Nesi, 1995] is a normalisation procedure over finite-state, serial, agents with guarded recursion. R_OBS is built upon STRAT, and, hence, suffers as well the problem of over-saturation. What is more, R_OBS, similar to some other class-rewriting systems, is prohibitively expensive, since rewriting modulo the equality over canonical systems of recursive equations [Bergstra and Klop, 1988], though computable, is impractically large. This bad response time makes R_OBS incompetent, when compared against state-space search based procedures. By comparison, `Observant` can be used to help reasoning about CCS agents regardless of their state space, as shown by [Monroy, 1996]; so, it can be incorporated to verification tools that can deal with problems outside the scope of state-space tools, such as the Edinburgh Concurrency Workbench (CWB) [Cleaveland *et al.*, 1989], the TAV system [Godskesen *et al.*, 1989], and the AUTO system [de Simone and Vergamini, 1989; Boudol *et al.*, 1990].

Annotated TRSs can be applied to guide the search for a solution. Often, the search space of ATRSs is smaller than those associated with non-annotated TRSs. ATRSs might be confluent, as in the case of `Observant`; but, more importantly, they give an account as to why, and how they achieve a task.

It is striking that those rules that belong to a class, e.g. attraction, saturation, etc. have some common structure. We reckon this is not coincidence; that similarity represents an operational idea, common to equational reasoning: e.g. to put all the occurrences of the unknown close together, or to isolate the unknown, etc. It would be nice to use learning techniques so as to exploit this similarity and produce a method for classifying and annotating the rules automatically.

References

[Bachmair and Plaisted, 1985] L. Bachmair and D. A. Plaisted. Termination orderings for associative-commutative rewriting systems. *Journal of Symbolic Computation*, 1:329–349, 1985.

[Basin and Walsh, 1996] David Basin and Toby Walsh. A calculus for and termination of rippling. *Journal of Automated Reasoning*, 16(1–2):147–180, 1996.

[Bergstra and Klop, 1988] J. A. Bergstra and J. W. Klop. A complete inference system for regular processes with silent moves. In *Logic Colloquium 86*, pages 21–81, Amsterdam, 1988. North Holland.

[Boudol *et al.*, 1990] G. Boudol, R. de Simone, V. Roy, and D. Vergamini. Process Calculi, from Theory to Practice: Verification Tools. In *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*. Springer-Verlag, 1990. Lecture Notes in Computer Science, v. 407.

[Bundy *et al.*, 1991] Alan Bundy, Frank van Harmelen, Jane Hesketh, and Alan Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991. Earlier version available from Edinburgh as DAI Research Paper No 413.

[Bundy *et al.*, 1993] Alan Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993. Also available from Edinburgh as DAI Research Paper No. 567.

[Bundy, 1988] Alan Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.

[Cleaveland *et al.*, 1989] R. Cleaveland, Parrow J., and B. Steffen. The concurrency workbench: A semantics-based verification tool for finite-state systems. In *Proceedings of the Workshop on Automated Verification Methods for Finite-State Systems*. Springer-Verlag, 1989.

[de Simone and Vergamini, 1989] R. de Simone and D. Vergamini. Aboard AUTO. Report 111, INRIA, 1989.

[Dershowitz, 1989] N. Dershowitz. Completion and its applications. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 31–86. Academic Press, 1989.

[Godskesen *et al.*, 1989] J. C. Godskesen, K. G. Larsen, and M. Zeeberg. Tav users manual. Internal report, Department of Computer Science, Aalborg University, 1989.

[Hennessy and Milner, 1985] M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the Association for Computing Machinery*, 32(1):137–161, 1985.

[Inverardi and Nesi, 1995] P. Inverardi and M. Nesi. Deciding observational congruence of finite-state ccs expressions by rewriting. *Theoretical Computer Science*, 139:315–354, 1995.

[Milner, 1989a] R. Milner. *Communication and Concurrency*. Prentice Hall, London, 1989.

[Milner, 1989b] R. Milner. A Complete Axiomatisation for Observational Congruence of Finite-State Agents. *Information and Computation*, 81:227–247, 1989. Also available from Edinburgh, as LFCS Report ECS-LFCS-86-8.

[Monroy *et al.*, 1994] R. Monroy, A. Bundy, and A. Ireland. Proof Plans for the Correction of False Conjectures. In F. Pfenning, editor, *5th International Conference on Logic Programming and Automated Reasoning, LPAR'94*, Lecture Notes in Artificial Intelligence, v. 822, pages 54–68, Kiev, Ukraine, 1994. Springer-Verlag. Also available from Edinburgh as DAI Research Paper 681.

[Monroy, 1996] R. Monroy. *Planning Proofs of Correctness of CCS Systems*. PhD thesis, Department of Artificial Intelligence, University of Edinburgh, Submitted to Faculty of Science and Engineering 1996.

[Park, 1981] D. Park. Concurrency and Automata on Infinite Sequences. In P. Deussen, editor, *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, 1981. LNCS 104.

[Sterling *et al.*, 1982] L. Sterling, Alan Bundy, L. Byrd, R. O'Keefe, and B. Silver. Solving symbolic equations with PRESS. In J. Calmet, editor, *Computer Algebra, Lecture Notes in Computer Science No. 144.*, pages 109–116. Springer Verlag, 1982. Also available from Edinburgh as DAI Research Paper 171.

[Walsh *et al.*, 1992] T. Walsh, A. Nunes, and A. Bundy. The use of proof plans to sum series. In D. Kapur, editor, *11th Conference on Automated Deduction*, pages 325–339. Springer Verlag, 1992. Lecture Notes in Computer Science No. 607. Also available from Edinburgh as DAI Research Paper 563.

[Yoshida *et al.*, 1994] Tetsuya Yoshida, Alan Bundy, Ian Green, Toby Walsh, and David Basin. Coloured rippling: An extension of a theorem proving heuristic. In A. G. Cohn, editor, *In proceedings of ECAI-94*, pages 85–89. John Wiley, 1994.